View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by UPCommons. Portal del coneixement obert de la UPC

Master's Thesis

Master's Degree in Industrial Engineering

Wi-Fi musical orchestra based on Raspberry PI

ANNEX

Author: Víctor Casanovas Ferrer
Director: Manuel Moreno Eguílaz
Call: April 2019

ETSEIB
Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Table of contents

1. Setting up SSH

Secure Shell (SSH) is a cryptographic network protocol for operating network services securely over an unsecured network **[1].** In this case it will be used to execute commands for the slave PIs remotely from the master PI.

In order to establish an SSH connection through a wireless network, the IP address associated with the slave PI must be known. It can be easily obtained through the *hostname -I* command If the slave Pi is headless, you have 2 options:

-Through your router devices list if you can access it.

-With the nmap command, which will ping the whole subnet range and will display a list of the devices that responded to it. The command is as follows:

$ nmap -sn 192.168.1.0/24

It will cover addresses 192.168.1.0 to 192.168.1.255.

Now if both PI's are connected to the same network and have SSH enabled, a connection can be established using the following command:

$ ssh pi@<IP>

Where IP is the IP of the slave.

If the connection was successful, commands entered through the current terminal will affect the Slave.

Additionally, if the flag -Y is added to the ssh command, it will allow remote interaction with graphical applications **[2].**

2. Setting up a Raspberry Pi as an access point in a standalone network (NAT)

Here are all the steps followed in order to use a Raspberry Pi with Raspbian 4.14 as an access point, obtained by combining two different guides, **[3]** and **[4].**

Install both DCHP server software and access point software and turn them off.

$ sudo apt-get install dnsmasq hostapd

$ sudo systemctl stop dnsmasq

$ sudo systemctl stop hostapd

Configure the static IP address.

$ sudo nano /etc/dhcpcd.conf

Add the following to the end of the file and restart.

interface wlan0

static ip_address=192.168.4.1/24

nohook wpa_supplicant

$ sudo service dhcpcd restart

Configure the DCHP server, rename the old config file and create a new one.

$ sudo mv /etc/dnsmasq.conf /etc/dnsmasq.conf.orig

$ sudo nano /etc/dnsmasq.conf

Add the following to *dnsmasq.conf*

interface=wlan0

listen-address=192.168.1.1

bind-interfaces

server=8.8.8.8

domain-needed

bogus-priv

dhcp-range=192.168.4.2,192.168.4.20,255.255.255.0,24h

Configure the access point host software.

$ sudo nano /etc/hostapd/hostapd.conf

// Add the following to dnsmasq.conf

interface=wlan0

driver=nl80211

ssid=RaspEtseib

hw_mode=g

channel=7

wmm_enabled=0

macaddr_acl=0

auth_algs=1

ignore_broadcast_ssid=0

wpa=2

wpa_passphrase=etseib123

wpa_key_mgmt=WPA-PSK

wpa_pairwise=TKIP

rsn_pairwise=CCMP

Tell the system where to find this configuration file.

$ sudo nano /etc/default/hostapd

Replace the line with #DAEMON_CONF with this.

DAEMON_CONF="/etc/hostapd/hostapd.conf"

Start the remaining services.

$ sudo systemctl start hostapd

$ sudo systemctl start dnsmasq

Add routing and masquerade, edit /etc/sysctl.conf and uncomment this line.

net.ipv4.ip_forward=1

$ sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE

$ sudo sh -c "iptables-save > /etc/iptables.ipv4.nat"

Edit /etc/rc.local and add this just above "exit 0" to install these rules on boot.

iptables-restore < /etc/iptables.ipv4.nat

Reboot and search for networks using a wireless device.

3. Code

3.1. Director.py

```
__author__ = 'Victor Casanovas Ferrer'
import socket
import struct
import time
import sys
from thread import *
import linecache
from threading import Thread
import threading
from ConfigParser import SafeConfigParser
from collections import deque

instruction = ''

class Director:
def __init__(self, parser):
      # Config parameters
      self.bpm = float(parser.get('song_params', 'bpm'))
      self.song_file_name = parser.get('song_params', 'song_file_name')
      self.instrument = parser.get('song_params', 'instrument')
      self.playbacks = parser.get('song_params', 'playbacks')
      self.mode = parser.get('song_params', 'mode')

      # Create the datagram sending socket
      self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

      # Set a timeout so the socket does not block indefinitely when trying
      to receive data.
      self.sock.settimeout(3)

      # Set the time-to-live for messages to 1 so they do not go past the
      local network
      segment.
      ttl = struct.pack('b', 2)
      self.sock.setsockopt(socket.IPPROTO_IP, socket.IP_MULTICAST_TTL, ttl)

      # Misc attributes
      self.note_list = []
      self.instr_address = []
      self.song_file_list = []
      self.time_to_wait_list = deque([])
      self.offset_list = deque([])
      self.note_interval = 6

# Get a list of all data files for a given song name
def list_files(self):
      for i in range(10):
            try:
                  open('Notes/' + self.song_file_name + str(i) + '.txt')
                  self.song_file_list.append('Notes/' + self.song_file_name
                  + str(i) + '.txt')
```

```
            except:
                  if self.song_file_list == []:
                        print >> sys.stderr, 'No files found with this
                        name.'
                        break
                  else:
                        print >> sys.stdout, '%s instruments are required
                        to play this song' % str(i)
                        break

# Get a list of all available instruments addresses, send instrument and
bpm data
def list_address(self):
      tries = 0
      multicast_group = ('224.3.29.72', 10000)
      while len(self.instr_address) < len(self.song_file_list):
            self.sock.sendto(str([self.bpm, self.instrument]),
            multicast_group)
            print >> sys.stdout, 'Detecting instruments'
            try:
                  data, address = self.sock.recvfrom(1024)
                  if data == 'init':
                        print >> sys.stdout, 'Connected to ', address
                        self.instr_address.append(address)
            except socket.timeout:
                  if tries >= 4:
                        print >> sys.stderr, 'Timed out, not enough
                        instruments detected'
                        break
                  else:
                        tries += 1

# Send a string to all addresses from the list and wait for acknowledgement
def send_to_all_and_verify(self, str, start_time):
      ack = 0
      tries = 0
      multicast_group = ('224.3.29.72', 10000)
      while ack < len(self.instr_address):
            seconds = (time.time() - start_time) % 60
            minutes = int((time.time() - start_time)/60)
            self.sock.sendto(str, multicast_group)
            print >> sys.stdout, '%02i:%06.3f - Sending %s to all
            instruments' % (minutes,
            seconds, str)
            try:
                  data, address = self.sock.recvfrom(1024)
                  if data == 'ack':
                        ack += 1
                        print >> sys.stdout, 'Received acknowledgement
                        from', address
            except socket.timeout:
                  if tries >= 4:
                        print >> sys.stderr, 'Timed out, not all
                        instruments responded'
                        break
                  else:
                        tries += 1
```

```
        return ack

# Get the maximum number of lines from a list of files
def send_file_len(self):
        j = 0
        for i in range(len(self.song_file_list)):
                with open(self.song_file_list[i]) as f:
                        for j, l in enumerate(f):
                                pass
                self.sock.sendto(str(j + 1), self.instr_address[i])

# Get a list of the next notes to play for each instrument
def update_notes(self):
        if len(self.note_list) == 0:
                for i in range(len(self.song_file_list)):
                        try:
                                f = open(self.song_file_list[i], 'r')
                                note = f.readline()
                                note = note[:-1]
                                note = note.split(' ')
                                for j in range(len(note)):
                                        fnote = float(note[j])
                                note[j] = float("%.3f" % fnote)
                                self.note_list.append(note)
                                self.note_list[i].append(2)
                        except:
                                print >> sys.stderr, "Couldn't find file " +
                                self.song_file_list[i]
                                break
        else:
                current_offset = self.min_offset()
                for i in range(len(self.song_file_list)):
                        if self.note_list[i][0] == current_offset and
                        self.note_list[i] != '':
                                try:
                                        note = linecache.getline
                                        (self.song_file_list[i], self.note_list[i][4])
                                        if note != '':
                                                note = note[:-1] # Remove /n
                                                note = note.split(' ')
                                                for j in range(len(note)):
                                                        fnote = float(note[j])
                                                        note[j] = float("%.3f" % fnote )
                                                self.note_list[i][:4] = note
                                                self.note_list[i][4] += 1
                                        else:
                                                self.note_list[i][0] = 9999
                                except:
                                        print >> sys.stderr, "Couldn't find file " +
                                        self.song_file_list[i]
                                        break
        return self.note_list

# Calculate the smallest offset in a list of notes
def min_offset(self):
        minoffset = min(self.note_list, key=lambda t: t[0])[0]
        return minoffset
```

```
# Send notes of minimum offset to their corresponding addresses
def send_notes(self, start_time):
      current_offset = self.min_offset()
      if len(self.instr_address) >= len(self.note_list):
            for i in range(len(self.note_list)):
                  if self.note_list[i][0] == current_offset:
                        data = self.note_list[i]
                        data = ['n'] + data + [float("%.3f" % sum(data))]
                        seconds = (time.time() - start_time) % 60
                        minutes = int((time.time() - start_time)/60)
                        self.sock.sendto(str(data), self.instr_address[i])
                        print >> sys.stdout, '%02i:%06.3f - Sending %s
                        to %s' % (minutes, seconds
                        , str(data), self.instr_address[i] )
      else:
            print >> sys.stderr, 'Timed out, not enough instruments
            detected'

# Send a string to all addresses from the list
def send_to_all(self, str, start_time):
      seconds = (time.time() - start_time) % 60
      minutes = int((time.time() - start_time)/60)
      self.sock.sendto(str, ('224.3.29.72', 10000))
      print >> sys.stdout, '%02i:%06.3f - Sending %s to all instruments' %
      (minutes, seconds, str)

def send_notes_buffer(self, start_time):
# Send notes until a buffer of 4 or more seconds is sent, then send a
buffer of 2
seconds every 2 seconds
      while self.min_offset() != 9999:
            offset_now = self.min_offset()
            offset_init = offset_now
            while self.offset_to_time(offset_now - offset_init, self.bpm)
            <= self.
            note_interval and offset_now != 9999:
                  self.send_notes(start_time)
                  self.offset_list.append(offset_now)
                  self.update_notes()
                  if self.min_offset() == 9999:
                        self.time_to_wait_list.append(0.15)
                        break
                  offset_next = self.min_offset()
                  self.time_to_wait_list.append(self.offset_to_time(offset_
                  next - offset_now,
                  self.bpm))
                  offset_now = offset_next
            self.note_interval = 2 # Define length of the group of notes to
            send in seconds
            time.sleep(self.note_interval)

def play_notes_buffer(self, start_time):
      correction = 0
      while len(self.time_to_wait_list) >= 1:
            init_time = time.time()
```

```
            self.send_to_all(str(['p', 1, self.time_to_wait_list[0],
            self.offset_list[0]]),start_time)
            time.sleep(self.time_to_wait_list[0]/2 - correction/2)
            self.send_to_all(str(['p', 0.5, self.time_to_wait_list[0],
            self.offset_list[0]]),start_time)
            time.sleep(self.time_to_wait_list[0]/2 - correction/2)
            self.send_to_all(str(['p', 0, self.time_to_wait_list[0],
            self.offset_list[0]]),start_time)
            self.offset_list.popleft()
            time_elapsed = self.time_to_wait_list.popleft()
            end_time = time.time()
            correction = (end_time - init_time) - time_elapsed

def offset_to_time(self, offset, bpm):
      time = offset * (15.0 / bpm)
      return time

def restart_attributes(self):
      self.note_list = []
      self.update_notes()
      self.note_interval = 6
      self.time_to_wait_list = deque([])
      self.offset_list = deque([])
      self.first = True
      self.hold = False

# Threaded function for user input
def user_input():
      global instruction
      while True:
            instruction = raw_input()
            time.sleep(2)


def main():

parser = SafeConfigParser()
parser.read('config.ini')

d = Director(parser)
d.list_files()

try:
      d.list_address() # Generate a list of instrument addresses
      time.sleep(0.25)
            if len(d.instr_address) == len(d.song_file_list) and
            d.song_file_list != []:
                  play = raw_input("Start playing the song? (y/n) \n")
                  # Wait for user confirmation
                  print_lock = threading.Lock()
                  if print_lock.acquire():
                        # Start a new thread for user input and return its
                        identifier
                        start_new_thread(user_input, ())
                  if play == 'y' and d.instr_address != []:
                        ack = d.send_to_all_and_verify('start',
                        time.time())
```

```
                        # Send start confirmation to all instruments and
                        await confirmation
                        time.sleep(0.25)
                        d.send_file_len() # Send file length point-to-point
                        to all instruments
                        for i in range(int(d.playbacks)):
                        # Repeat as many times as stated in config file
                              time.sleep(0.25)
                              print >> sys.stdout, 'Playback number ' +
                              str(i + 1)
                              d.restart_attributes()
                              offset1 = d.min_offset()
                              start_time = time.time()
                              print(d.note_list)
                              if d.mode == 'buffer':
                                    notes_thread =
                              Thread(target=d.send_notes_buffer,
                                    args=(start_time,))
                                    notes_thread.daemon = True
                                    notes_thread.start()
                                    time.sleep(0.25)
                              # Main loop, repeat if not all notes have been
                              sent and all instruments received start
                              confirmation
                              while d.min_offset() < 9999 and ack ==
                              len(d.instr_address):
                              # Check for user input
                              while instruction == 'pause' or instruction ==
                              'stop' or   d.hold:
                                    d.hold = True
                                    time.sleep(1)
                                    start_time += 1
                                    if instruction == 'stop':
                                          start_time = time.time()
                                          d.first = True
                                          d.update_notes()
                                          offset1 = d.min_offset()
                                    if instruction == 'start':
                                          d.hold = False
                                          break
                              if d.mode == 'buffer':
                                    d.play_notes_buffer(start_time)

except KeyboardInterrupt:
      print >> sys.stderr, 'Keyboard interrupt'
except:
      print >> sys.stderr, 'Unexpected error'
finally:
      print >> sys.stdout, 'Closing application'

d.sock.close()
      if __name__ == "__main__":
main()
```

3.2. Instrument.py

```
__author__ = 'Victor Casanovas Ferrer'
import socket
import struct
import sys
import time
from threading import Thread
from collections import deque

class Instrument:
def __init__(self):
      self.fs = 0
      self.bpm = 60
      self.instrument = 'example.sf2'
      self.notes_played = 0
      self.notes_played_buffer = 0
      self.notes_skipped = []
      self.bad_checksum = 0
      self.note_len = 0
      self.note_index = 1
      self.start_time = time.time()
      self.wait_time = 0
      self.mode = 'wait_for_start'
      self.note_list = deque([])
      self.note_played_time = 0
      self.upper_margin = 0
      self.lower_margin = 0
      self.predict_margin = 0.02
      self.process_time = 0

      # Create the datagram receiving socket
      self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
      self.sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
      # Set a timeout so the socket does not block indefinitely when trying
      to receive data.
      self.sock.settimeout(60)
      # Bind to the device address
      self.sock.bind(('', 10000))
      # Tell the operating system to add the socket to the multicast group
      on all
      interfaces.
      group = socket.inet_aton('224.3.29.72')
      mreq = struct.pack('4sL', group, socket.INADDR_ANY)
      self.sock.setsockopt(socket.IPPROTO_IP, socket.IP_ADD_MEMBERSHIP,
      mreq)

# Receive bpm and instrument data from director and respond
def detect_director(self):

      while True:
            try:
                  print >>sys.stdout, 'Looking for Director'
                  data, address = self.sock.recvfrom(128)
                  self.bpm = float(eval(data)[0])
                  self.instrument = eval(data)[1]
```

```
                print >>sys.stdout, 'Received "%s" from "%s,
            responding..."' %       (data, address)
                self.sock.sendto('init', address)
                break

            except socket.timeout:
                print >>sys.stderr, 'Timed out, no response'
                break

            except:
                print >>sys.stderr, 'Incorrect data'

# Load soundfont file and create fluidsynth object
def load_fluidsynth(self):

    try:
        import fluidsynth
        # Fludisynth setup
        self.fs = fluidsynth.Synth()
        self.fs.start()

        # Load soundfont file
        try:
            self.start_time = time.time()
            sfid = self.fs.sfload(self.instrument)
            self.fs.program_select(0, sfid, 0, 0)
            time_elapsed = time.time() - self.start_time
            print >>sys.stderr, 'SoundFont file loaded in %05.2f' %
            time_elapsed
        except:
            print >>sys.stderr, 'Could not load SoundFont file'
    except ImportError:
        print >>sys.stderr, 'Could not find PyFluidsynth module'


def play_note(self, note):
    if self.fs != 0:
        time.sleep(self.wait_time)
        self.fs.noteon(0, int(note[2]), int(note[3]))
        time.sleep((note[4]*15/self.bpm))
        self.fs.noteoff(0, int(note[2]))
    else: # Executed from desktop
        time.sleep(self.wait_time)
        current_time = (time.time() - self.start_time) % 60
        minutes = int((time.time() - self.start_time)/60)
        print >>sys.stderr, '%02i:%06.3f - Note %s ON' % (minutes,
        current_time, str(note))
        time.sleep((note[4]*15/self.bpm))
        current_time = (time.time() - self.start_time) % 60
        minutes = int((time.time() - self.start_time)/60)
        print >>sys.stderr, '%02i:%06.3f - Note %s OFF' % (minutes,
        current_time, str(note))

def play_from_buffer(self, note, time_for_next_note, margin):
    time.sleep(time_for_next_note + margin - self.process_time)
    # Wait expected time +30ms
    start_time = time.time()
    if (time.time() - self.note_played_time) > time_for_next_note:
```

```
# Time since last note played from a play order > time expected for
next note to be played
      if len(self.note_list) > 1: # 2 or more notes to play
            # If previous notes have not been played yet, remove them
            from the list
            while note[1] > self.note_list[0][1]:
                  print >>sys.stdout, 'Removed note %s from list
                  while playing from buffer' % str(self.note_list[0])
                  self.note_list.popleft()
            if self.note_list[0] == note: # Note not played yet
                  delta_offset = (self.note_list[1][1] -
                  self.note_list[0][1])*15/self.bpm
                  self.process_time = time.time() - start_time
                  # Recalculate times at which the next note should
                  be played
                  if margin == 0.03:
                        self.upper_margin = time.time() -
                        self.process_time + delta_offset +
                        self.predict_margin - margin
                        self.lower_margin = time.time() -
                        self.process_time + delta_offset -
                        self.predict_margin - margin
                  else:
                        self.upper_margin = time.time() -
                        self.process_time + delta_offset +
                        self.predict_margin
                        self.lower_margin = time.time() -
                        self.process_time + delta_offset -
                        self.predict_margin

                        # Play note from buffer
                        self.wait_time = 0
                        t3 =
                        Thread(target=self.play_note,args=(note,))
                        t3.start()
                        current_time = (time.time() -
                        self.start_time) % 60
                        minutes = int((time.time() -
                        self.start_time)/60)
                        print >>sys.stdout, "%02i:%06.3f - Playing
                        note %s from buffer" % (minutes, current_time,
                        str(note))

                        # Update note list and counters
                        next_note = self.note_list[1]
                        time_for_note = (self.note_list[1][1] -
                        self.note_list[0][1])*15/self.bpm
                        self.note_list.popleft()
                        self.notes_played += 1
                        self.notes_played_buffer += 1
                        self.process_time = start_time - time.time()

                        # Compensate margin if we still play from
                        buffer and attempt to play
                        next note from buffer
                        if margin == 0.03:
```

```
                                    self.play_from_buffer(next_note,
                                    time_for_note, -0.03)
                              else:
                                    self.play_from_buffer(next_note,
                                    time_for_note, 0)
                        else:
                              print >>sys.stdout, "Note %s already played,
                        next note: %s" %  (str(note),str(self.note_list[0]))

def store_results(self):
      # Store number of notes played
      f = open("Performance_data.txt","a")
      f.write(str(self.notes_played) + "\n")
      f.close()
      print >>sys.stdout, 'Received and played %s out of %s notes, %s from
      buffer' % (str(
      self.notes_played), str(self.note_len),
      str(self.notes_played_buffer))

      # Store list of notes skipped
      if len(self.notes_skipped) == 0:
            print >>sys.stdout, 'No notes skipped'
      else:
            f = open("Notes_skipped_data.txt", "a")
            for i in range(len(self.notes_skipped)):
                  f.write(str(self.notes_skipped[i]) + ",")
            f.write("\n")
            f.close()
            print >>sys.stdout, 'Skipped
            notes %s' %(str(self.notes_skipped))

# Store the length of the note list
def store_buffer_size(self):
      if len(self.note_list) > 1:
            buffer_size = (self.note_list[-1][1] -
            self.note_list[0][1])*15/self.bpm
            current_time = (time.time() - self.start_time) % 60
            minutes = int((time.time() - self.start_time)/60)
            line = '%02i:%06.3f, %f' % (minutes, current_time, buffer_size)
            f = open("Buffer_size.txt", "a")
            f.write(line + "\n")

def main():
      note = []
      ins = Instrument()
      ins.detect_director()
      ins.load_fluidsynth()
      while ins.mode != 'end':
            try:
                  ins.data, ins.address = ins.sock.recvfrom(64)
            except socket.timeout:
                  print >>sys.stderr, 'Warning: Timed out, no response'
                  break
            else:
                  if ins.data == 'end':
                        ins.mode = 'end'
                        print >>sys.stdout, 'Ending'
```

```
                ins.sock.sendto('ack', ins.address)
        elif ins.mode == 'wait_for_start':
                if ins.data == 'start':
                        print >>sys.stdout, 'Received start
                        confirmation'
                        ins.sock.sendto('ack', ins.address)
                        ins.start_time = time.time() + 0.5
                        ins.mode = 'wait_for_len'
                else:
                        print >>sys.stderr, 'Warning: Expected start
                        confirmation'
        elif ins.mode == 'wait_for_len':
                try:
                        if ins.data == 'start':
                                pass
                        else:
                                ins.data = int(ins.data)
                                ins.note_len = ins.data
                                ins.mode = 'wait_for_note_or_play'
                                print >>sys.stdout, 'Notes to
                                play: %s' % (ins.data)
                except ValueError:
                        print >>sys.stderr, 'Warning: Expected note
                        list length, skipping'
                        ins.mode = 'wait_for_note_or_play'
        elif ins.mode == 'wait_for_note_or_play':
                ins.list = eval(ins.data)

                if ins.list[0] == 'n': # Note received
                        if ins.list[6] == sum(ins.list[1:6]):
                        # Checksum ok
                                if ins.note_index + 1 == ins.list[5] or
                                ins.list[5] == 2:
                                # Check continuity on notes
                                        ins.note_list.append(ins.list)
                                        current_time = (time.time() -
                                        ins.start_time) % 60
                                        minutes = int((time.time() -
                                        ins.start_time)/60)
                                        print >>sys.stdout, '%02i:%06.3f -
                                        Received "%s" from %s' % (
                                        minutes, current_time, ins.data,
                                        ins.address)
                                else:
                                        for i in range(ins.list[5] -
                                        ins.note_index):
                                        ins.notes_skipped =
                                        ins.notes_skipped +
                                        [ins.note_index + i +1]
                                        print >>sys.stderr, 'Warning:
                                        Discontinuity receiving notes'
                                ins.note_index = ins.list[5] # Update
                                index
                        else:
                                print >>sys.stderr, 'Warning: Checksum
                                incorrect'
                                ins.bad_checksum += 1
```

```
elif ins.list[0] == 'p': # Play order received
      current_time = (time.time() - ins.start_time) % 60
      minutes = int((time.time() - ins.start_time)/60)
      print >>sys.stderr, "%02i:%06.3f - Received PO %s" %
      (minutes,
      current_time, str(ins.list))
      # There are notes to play
      if ins.note_list != deque([]):
      # If PO offset is equal to the offset of the first note
      in the queue
            if ins.list[3] == ins.note_list[0][1]:
                  ins.wait_time = ins.list[1] * ins.list[2]
                  # If PO play note time is within predicted
                  range
                  if ins.lower_margin <= time.time() +
                  ins.wait_time < ins.
                        upper_margin or ins.notes_played == 0:
                              if len(ins.note_list) > 1:
                              # Attempt to play next note
                              from buffer
                                    next_note = ins.note_list[1]
                                    time_for_next_note =
                                    ins.wait_time +
                                    (ins.note_list[1][1]
                                    -ins.note_list[0][1])*15/
                                    ins.bpm
                                    t1 =
                                    Thread(target=ins.play_from_
                                    buffer, args=(next_note,
                                    time_for_next_note, 0.03,))
                                    t1.daemon = True
                                    t1.start()

                                    # Recalculate times at which
                                    the next note should be
                                    played
                                    ins.upper_margin =
                                    time.time() + ins.wait_time
                                    + (ins.note_list[1][1] -
                                    ins.note_list[0][1])*15/ins.
                                    bpm + ins.predict_margin

                                    ins.lower_margin =
                                    time.time() + ins.wait_time
                                    + (ins.note_list[1][1] -
                                    ins.note_list[0][1])*15/ins.
                                    bpm - ins.predict_margin

                              # Play current note
                              note = ins.note_list.popleft()
                              ins.note_played_time = time.time()
                              t2 = Thread(target=ins.play_note,
                              args=(note,))
                              t2.start()
                              ins.notes_played += 1
```

```
                                    # If it is the last note of the
                                    playback, restart attributes
                                    if note[5] == ins.note_len + 1:
                                          ins.store_results()
                                          ins.notes_played = 0
                                          ins.notes_skipped = []
                                          ins.note_index = 1
                                          ins.note_list = deque([])
                              else:
                                          current_time = (time.time() -
                                          ins.start_time) % 60
                                          minutes = int((time.time() -
                                          ins.start_time)/60)
                                          print >>sys.stderr, "%02i:%06.3f -
                                          Warning: mistimed play
                                          order Upper - PO = %s" % (minutes,
                                          current_time, ins.upper_margin -
                                          (time.time() + ins.wait_time))
                        else:
                                    print >>sys.stderr, "Warning: note list empty"
                  else:
                        print >>sys.stderr, 'Warning: Expected play order
                        or note data, received %s' % ins.data

      print >>sys.stdout, 'Closing socket'

      ins.sock.close()
      print >>sys.stdout, 'Waiting for all threads to turn off notes'

      try:
            time.sleep(1.5*(note[4]*15/ins.bpm))
      except:
            pass
      if ins.fs != 0:
            ins.fs.delete()
      if __name__ == "__main__":
            main()
```

3.3. Partiture.py

```
# Generate partitures from midi file
# Manuel Moreno
# October 2015

from music21 import *
import time

class Partiture():
      def __init__(self, midifile):
            self.name = midifile
            #Read midi file & get information
            mf = midi.MidiFile()
                  mf.open(midifile+'.mid')
            mf.read()
```

```
            mf.close()
            self.s = midi.translate.midiFileToStream(mf)
            self.tracks = len(self.s.parts)
            print("Tracks:",self.tracks)
            for i in range(self.tracks):
                  print("Len:",len(self.s.parts[i].flat.notes))
            self.t = [[-1 for x in range(400)] for x in range(self.tracks)]
#time
            self.n = [[-1 for x in range(400)] for x in range(self.tracks)]
#note
            self.v = [[-1 for x in range(400)] for x in range(self.tracks)]
#velocity
            self.d = [[-1 for x in range(400)] for x in range(self.tracks)]
#duration

def go(self):
      for i in range(self.tracks):
            with open(self.name+str(i)+'.txt', "w") as f:
#open file midifilename + track index + txt

                  for j in range(len(self.s.parts[i].flat.notes)):
                        try:
                              if self.s.parts[i].flat.notes[j].isChord ==
False:
                                    mistr
=str(float(self.s.parts[i].flat.notes[j].offset))+' '+ str(self.s.parts[i
].flat.notes[j].pitch.midi)+ ' ' +str(self.s.parts[i].flat.notes[j].volume.
velocity)+ ' ' + str(self.s.parts[i].flat.notes[j].duration.
quarterLengthNoTuplets)
                                    f.write(mistr+'\n')
                              else:
                                    print("Chord:",i,j)
                        except:
                              pass
            print("Read Over")

def read(self):
      t1 = time.time()
      for i in range(self.tracks):
            j = 0
            with open(self.name+str(i)+'.txt') as f:
                  for line in f:
                        data = line.split()
                        self.t[i][j] = float(data[0])
                        self.n[i][j] = int(data[1])
                        self.v[i][j] = int(data[2])
                        self.d[i][j] = float(data[3])
                        j = j + 1
      t2 = time.time()
      print(t1,t2)
      print(t2-t1)


if __name__ == '__main__':
      par1 = Partiture('The_Piano') #('vival')#Canon2')
      par1.go()
      par1.read()
```

Bibliography

Bibliographic references

[1] SSH. *"SSH (Secure Shell)".* (Cons. 12/09/2018).
URL: https://www.ssh.com/ssh/

[2] Raspberry Pi. *"SSH using Linux or Mac OS"* (Cons. 12/09/2018).

URL: https://www.raspberrypi.org/documentation/remote-access/ssh/unix.md

[3] Raspberry Pi. *"Setting up a Raspberry Pi as an access point in a standalone network (NAT)"* (Cons. 18/07/2018).

URL: https://www.raspberrypi.org/documentation/configuration/wireless/access-point.md

[4] Alejandro Antón Turc. *"Millores d'un micro-braç articulat*" (Annex, Pags. 22-24)